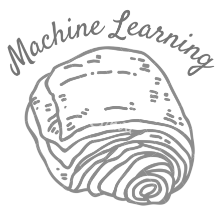
CentraleSupélec

Kaggle Team :
Pain au chocolat



Kaggle project report

Students:
DENG Erwin
AL BOUZIDI Elias
ALAMI Nabil

Teacher:
Nora Ouzir

TA:
Aaron Mamann, Mohammad Reza Mirafzal, Hiba Hojeij, Marouane Battach

September 15, 2025

1 Feature engineering

Our dataset contains 296146 training data, 120526 test data, and 44 features (numerical, categorical, dates, and geometrical). The first step is to process our data to extract key information. This step is essential to have great performance at the end.

1.1 Missing values

Our dataset contains 4249 rows with missing values (dates or image color). This doesn't seem to be a huge issue since it represents only 1% of our data. We could either drop (only for train data) or impute them. For the training set, we decided to **impute the missing values**, by replacing the missing values with the most frequent value among the data with the same `change_type`. For the test set, we decided either to let them missing, or imputing them with the median value of all the dataset.

1.2 Column: urban_type

`urban_type` is a categorical feature which contains a list of all urban types. Given the small number of urban types (7), and given the fact that the labels can't be ordered, we decided to do **one-hot encoding**, by creating 7 columns for each urban types. We also grouped the columns `urban_type_N` and `urban_type_A`, since `N,A` probably means that the value is missing. We also added one-hot-encoding for mix of urban types, since there are only 4 mixes observed in the training set.

1.3 Column: geography_type

`geography_type` is a categorical feature which contains a list of all geography types. We'll do the same thing as `urban_type`: given the small number of geography types (13) and given the fact that the labels can't be ordered, we decided to perform **one-hot encoding**. We also grouped the columns `geography_type_N` and `geography_type_A`. This time, we won't create one-hot encoding for all mixes of geography_type because there are many mixes observed in the training set. We'll only create them for mixes that contains more that 3900 values (which corresponds to the 20st most encountered geography_type)

1.4 Column: geo_urban

We could create columns for mixes of `geography_type` and `urban_type`. We decided to do this only for mixes which contains at least 1% of a `change_type`. We created 34 columns (instead of $6 \times 12 = 72$ columns).

1.5 Renaming and Sorting dates

The columns `dates` and `change_status` are numbered from 0 to 4 while the columns related to `img` are numbered from 1 to 5. Thus, we renamed everything to 0 to 4. Moreover, the columns `date0`, `date1`, ... `date4` are not sorted! Thus, we reorganized all the time-related columns according to the chronological order.

1.6 Column: change_status

`change_status` gives the status of the building at each date. It is a categorical variable that contains 10 different values. We created dummy variables for each label and each date: it creates 50 features (ex: `change_status_date4_Greenland`).
We can also group all the columns regardless of the dates (ie we compute the sum of each label over all the 5 dates). It creates 10 new features (ex: `change_status_Greenland`).
We also though about creating a column that count each possible transition. But since it creates $10 \times 10 = 100$ features, we abandonned let this idea.
But the labels can be ordered: following the article on the dataset, we chose the following order:
"Prior Construction": 0, "Greenland": 1, "Land Cleared": 2, "Excavation": 3, "Materials Dumped": 4, "Materials Introduced": 5, "Construction Started": 6, "Construction Midway": 7, "Construction Done": 8, "Operational": 9.
We can thus encode the change_status, thus creating 5 new features (instead of 50 with one-hot encoding).
Finally, we can compute the variations (for example, `change_status_date4 - change_status_date0`). We did it for 4-0, 1-0, 2-1, 3-2, 4-3, thus creating 5 new features.

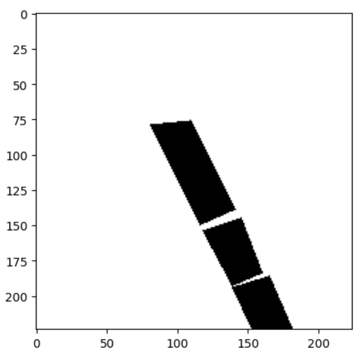
1.7 Column: colors

For the color-related features, we decided to compute the mean of the 3 colors, thus creating 10 new features
ex: `img_mean_date0` = (`img_red_mean_date0`+`img_blue_mean_date0`+`img_green_mean_date0`)/3
We also computed the differences between dates 4-0, 1-0, 2-1, 3-2, 4-3.

1.8 Column: dates

For the dates, it might be not usefull to keep the dates in a string format. We computed the difference in days between 4-0, 1-0, 2-1, 3-2, 4-3.
Besides, we also computed the difference in days between the date0 and the minimum of the the date0. Note that this might give overfitting due to the fact that it has no link with the result, unless our test set is similar to our train set.
And finally, we created new features by dividing the differences computed before by the date (for img and change_status).

1.9 Column: geometry

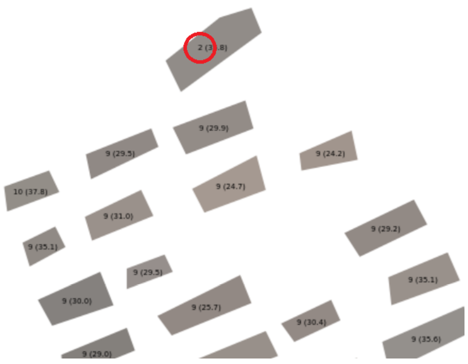


For each building, we have access to a polygon. Thus, we computed many geometric features: area, perimeter, inscribed circle radius, compactness, convexity, number of vertices, bbox width (max difference in x), bbox height, bbox ratio, bbox area, bbox perimeter, diameter, minimum bounding circle, and minimum rotated rectangle.

But for each building, the polygon is represented by its latitude and longitude. Thus, it is actually possible to exploit the relation between the surrounding of each building.
For each building, we plotted, for a given scale, the shape of the building and the building nearby (see the figure on the left). We thus computed: `neighb_density` the mean of the whole image, `neighb_nb` the number of building in the image and `neighb_similarity` the mean hausdorff distance between the current building and the building around, that tells us how similar the current building is from the other buildings.

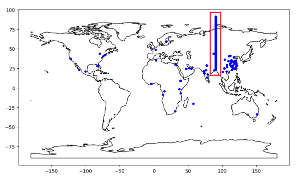
We also computed for each building :

- the maximum difference in change_status with its surrouding (ex: `neighb_maxdiff_change_status_date0_0.003` where 0.003 is the maximum radius for the surrounding)
- the maximum difference in absolute value for the color mean and std for each date (ex: `neighb_maxdiff_meancolor_0`)
- the difference between the current building and the mean of the surrounding for change_status, color mean and std.



This intuition comes from several observation: we see that, given a certain context, the most important is not the color in itself, but the color difference with its surrounding. Here, the value circled in red is the change_status, which is very different from its surrounding.

Besides, by ploting the points on a map (that is used just for observation), we see that there is a line of 1850 weird points. By having a closer look, we realise that they are rectangles of hundred of kilometers. Since they are both in the train and test set, we decided to remove them during training.
Finally, we also tried to use a CNN pretrained with Resnet to extract key features about the shape of each building.



1.10 What features we should keep

The CNN performed really badly (decreasing f1-score by 10%). Thus, we decided to discard these features. Moreover, we decided not to use the latitude and longitude of each polygon since it would lead to bad generalisation for unseen data.
All the other 335 features where kept, since we'll use mostly decision trees that perform feature selction. However, we noticed that geometric features, features starting with `neighb_` and the change_status were the most important features, increasing the f1-score in all models. Dates also improve our score. Image color also improve our score. Geography and urban doesn't improve that much our score.

2 Performing Models

For all models, we split our data in training (80%) and validation sets (20%). Thus, we can compare different methods with the validation set, and be sure to avoid overfitting. If the computation time isn't too long, we use cross validation with K-fold (K=5, stratified).

2.1 XGBoost

Boosting algorithms have become one of the most powerful algorithms for training on tabular data, and XGBoost is one of them. We fix `learing_rate` to a reasonable value (between 0.05 and 0.3), and we choose a large `n_estimators` so that our score on validation converges. To prevent overfitting, we use `early_stopping_rounds` to stop the training when the validation score doesn't increase anymore. Then, we fine-tune the other parameters: `lambda`, `alpha`, `gamma`, `colsample_bytree`, `subsample`, `max_depth` and `min_child_weight`. These parameters allows to find a tradeoff between the training score and overfitting. We can also modify `sample_weights` in order to take into account (or not) the unbalanced dataset. We performed this tuning manually first, and with the library `optuna` which test and select automatically the best parameters. After training a model, it is also possible to look at the most important features, and eliminate them if it creates overfitting. Thus, we identified that `date0`, the latitude and longitude were important, but encourage overfitting. Thus, we removed them. When the best parameters are found, we reduce the learning rate, find the ideal number of estimators (until early_stopping stops our training), and train on the whole training set. Here are 3 models with good scores on validation set:

XGBoost_13: `max_depth=13`, `colsample_bytree = 0.3`, `subsample = 1.0`, `min_child_weight=29`, `gamma=0.00318`, `lambda=0.373`, `alpha=0.0135`. $\boxed{\text{Val score 0.790 | Kaggle score 0.97602}}$

XGBoost_17: same parameters with `max_depth=17`. $\boxed{\text{Val score 0.790 | Kaggle score 0.97498}}$

XGBoost_15: `max_depth=15`, `sample_weight=classes_weights`, `colsample_bytree=0.7`, `gamma=7` $\boxed{\text{Val score 0.778 | Kaggle score 0.97588}}$

2.2 CatBoost

CatBoost is also a boosting algorithm, which is quite similar to XGBoost. It was developed by Yandex in 2017. We followed the same methodology for fine tuning (since the parameters are really similar). We obtained:

CatBoost: `l2_leaf_reg=6`, `depth=12`, `bagging_temperature=0`, `max_bin=512`. $\boxed{\text{Val score 0.785 | Kaggle score 0.97498}}$

2.3 LightGBM

LightGBM is another boosting algorithm developped by Microsoft. We didn't have time to fine tune it very well, but we obtained:

LightGBM: $\boxed{\text{Val score 0.775 | Kaggle score 0.97448}}$

2.4 RandomForest

We also tried using RandomForest, which is commonly used for multiclassification. First, we fix a low number of estimators (200) in order to compute quickly many combinations of parameters. Then, we try many combinations (manually, with gridsearchcv and with optuna) of `max_depth`, `min_samples_split`, `min_samples_leaf` and `class_weight`. We obtain:

RandomForest: `n_estimators=1000`, `max_depth=73`, `min_samples_split = 8`, `min_samples_leaf = 2`, `class_weight="balanced_subsample"`. $\boxed{\text{Val score 0.7718 | Kaggle score 0.9726}}$ RandomForest allowed us to visualize the importance of each feature, and as expected, the geometric features and $t_4 - t_1$ were the most important factors.

2.5 KNN

KNN is a really simple model. But with normalization, and using $k = 5$, `weights=distance`, we get $\boxed{\text{Val score 0.70}}$ which is not bad! Actually, we'll do a simple trick which consists of using a 1-NN and keeping the nodes at distance 0: when a test set data is also contained in the training set, the 1NN will give the correct response all the time. We had this idea when we saw that our Kaggle score was way better than our validation score: indeed, 88533 tests data (over 120526) are actually already in the training set! This method gives $\boxed{\text{Kaggle score 0.81114}}$.

2.6 Other attempts

We tried to reduce dimensions with $\boxed{\textbf{PCA and SVD}}$. But theses methods didn't allow us to obtain better results, given the fact that boosting algorithm already deal with feature selection.

We implemented $\boxed{\textbf{SVM}}$ with an RBF kernel. Even though SVMs are natively designed for binary classification, we can use SVM for multiclass problems following the One-to-One approach. The results were very poor: 0.31 score at Kaggle. We didn't try to optimize it since its computation time grows rapidly with the size of the dataset and supported vectors (more than 300 minutes for training and 40 minutes for predictions).

We thought that maybe drawing again all the polygons and using a $\boxed{\textbf{CNN}}$ might work, and create more features. We created a CNN combined with a MLP, but it gave bad results for a long computation time (validation score: 0.69)

2.7 Final model: Vote

We use our best models : XG_Boost_13, XG_Boost_15, XG_Boost_17, LightGBM, and CatBoost, and combine them by a system of vote. We assign a coefficient to each model, and we use either their probability for each class (weighted sum of each class), or directly the class predicted. Finally, we replace the predicted values by the 1-NN.

CatBoost	LightGBM	XGBoost13	XGBoost17	XGBoost15	Vote	Kaggle result
4	1	5	3	4	Class	0.97611
4	1	4	2	4	Class	0.97604
0	0	1	0	0	Class	0.97604
1	1	4	1	4	Probs	0.97598
3	0	7	2	4	Probs	0.97592